# ESP32-S3-LCD-1.47B Board Analysis

**Executive Summary:** The Waveshare **ESP32-S3-LCD-1.47B** (Type B) is an HMI-focused development board built around the Espressif ESP32-S3 series SoC (Xtensa LX7 dual-core, up to 240 MHz). It integrates a 1.47″ IPS TFT display (172×320, ST7789 controller), 16 MB SPI flash, 8 MB QSPI PSRAM, a 6-axis IMU (QMI8658), RGB LED (WS2812B), TF card slot, USB-Type-C port, and Li-ion charger. Key features include 2.4 GHz Wi-Fi (802.11b/g/n), Bluetooth 5.0, 512 KB SRAM, 384 KB ROM, plus a versatile GPIO header for additional sensors/peripherals[1]. Power is supplied via USB-C or a LiPo battery (with on-board charging, supported by a ME6217C33M5G LDO and charger IC). This report provides a detailed engineering dossier: a **system block diagram**, full GPIO pin mapping, component overview, risk analysis (power, thermal, signal, firmware/boot), test plans, firmware boot flow, and recommendations.

## System Architecture

The board's **functional blocks** are:

- **MCU:** ESP32-S3 SoC (Xtensa LX7 dual-core, 240 MHz).
- **Storage:** 16 MB SPI Flash, 8 MB QSPI PSRAM (external) for program and graphics data.
- **Display Subsystem:** 1.47″ IPS TFT (ST7789 driver) connected via SPI, plus backlight LED controlled via GPIO.
- **Sensors/IO:** On-board QMI8658 IMU (I²C-connected 3-axis gyro+accelerometer), WS2812 RGB LED (GPIO-driven), BOOT/RESET buttons, and a multi-pin expansion header for digital/analog I/O (UART, SPI, I²C, PWM, etc.).
- **Power System:** USB-C port (VBUS), a battery charging circuit with LiPo connector, 3.3 V LDO regulator (ME6217C33M5G) feeding 3V3 rail, plus 5 V, GND rails.
- **USB Interface:** On-chip USB-Serial JTAG peripheral (USB_D+, USB_D-) for programming/console via Type-C connector.

A simplified block diagram (logical interconnections) is shown below:

```
flowchart LR
    subgraph MCU_Block [ESP32-S3 MCU]
        MCU[ESP32-S3 MCU (LX7 dual-core)]
    end
    MCU -->|SPI / QSPI| ST7789[1.47″ TFT (ST7789)]
    MCU -->|SPI / QSPI| Flash16MB[16 MB SPI Flash]
    MCU -->|QSPI| PSRAM8MB[8 MB PSRAM]
    MCU -->|USB (DP/DN)| USB_IFC[USB Type-C Port (USB OTG)]
    MCU -->|GPIO/I²C| IMU[6-axis IMU (QMI8658)]
```

```
MCU -->|GPIO (one-wire)| RGB_LED[WS2812 RGB LED]
MCU -->|GPIO| Buttons[BOOT/RESET Buttons]
MCU -->|SPI/I²C/UART/...| Headers[GPIO Expansion Header]
USB_IFC -->|5 V| Charger[Battery Charger IC]
Charger -->|VBAT| Battery[LiPo Battery (3.7 V)]
Battery -->|3.7 V| Charger
Charger -->|3.3 V| LDO[ME6217 3.3 V LDO]
LDO -->|3.3 V| MCU
Headers -->|Sensors| External[External Devices/Sensors]
```

- The **MCU** block (ESP32-S3) drives all peripherals. It interfaces with the LCD via an SPI-like parallel interface (specifically, the ST7789 uses a variant of SPI with DC and CS signals). It also controls the backlight LED via a MOSFET (Q1) and supports I²C/SPI/UART on the header.
- **Flash** and **PSRAM** are both SPI/QSPI chips connected to the SoC's flash interface (pins SPID, SPIQ, etc.).
- The **USB-C port** provides power (5 V) and connects to the ESP32-S3's native USB-Serial function (USB_DP, USB_DM) for programming. This port also feeds the battery charger IC.
- The **battery charger** manages a 3.7 V LiPo and provides 3.3 V (via ME6217 LDO) to the board.
- The **IMU** (QMI8658) is connected via I²C to detect motion/gesture. The **WS2812 LED** is driven by a single GPIO (RMT/PWM).
- **Buttons:** One is the **BOOT** button (tied to GPIO0 strap) and one is the **RESET** (EN/CHIP_PU pin) button.

This arrangement allows the ESP32-S3 to run rich GUI code (e.g. using LVGL) while simultaneously handling wireless, sensors, and storage.

## Pin Mapping Matrix

Below is a **GPIO pin matrix** (ESP32-S3 signals) for the board. Columns list the GPIO name, SoC pin name, its main function, what it's wired to, and any special notes or constraints (strap pins, pull-ups/downs, high-speed signal, voltage limits, etc.):

| GPIO | SoC Pin | Function | Connected To | Notes/ Constraints |
|---|---|---|---|---|
| GPIO0 | IO0 (Chip 5) | BOOT (strapping) | BOOT Button (to GND when pressed) | Pull-up via 10 kΩ to Vcc; hold low at reset to enter serial bootloader |

| GPIO | SoC Pin | Function | Connected To | Notes/ Constraints |
|---|---|---|---|---|
| | | | | [2]. |
| GPIO1 | U0TXD (49) | UART0_TXD | (USB-serial, not broken out externally) | Default UART TX. Not used as header I/O (goes to USB CDC). |
| GPIO3 | U0RXD (50) | UART0_RXD | (USB-serial, not broken out externally) | Default UART RX. Not used as header I/O. |
| GPIO14 | IO14 (23) | SD_SCK (SDIO CLK) | TF Card SD_SCK (CLK) | SDIO clock line to TF slot; supports up to ~40 MHz[3]. Pull-ups present (10 kΩ on CMD/Dx lines). |
| GPIO15 | IO15 (26) | SD_CMD (SDIO CMD) | TF Card SD_CMD (CMD) | SDIO command line; internal pull-ups on card (per schematic). |
| GPIO16 | IO16 (27) | SD_D0 (SDIO D0) | TF Card SD_D0 (D0) | SDIO data 0. |
| GPIO17 | IO17 (23) | SD_D2 (SDIO D2) | TF Card SD_D2 (D2) | SDIO data 2. |
| GPIO18 | IO18 (24) | SD_D1 (SDIO D1) | TF Card SD_D1 | SDIO data 1. |

| GPIO | SoC Pin | Function | Connected To | Notes/ Constraints |
|---|---|---|---|---|
| | | | (D1) | |
| GPIO21 | IO21 (27) | SD_D3 (SDIO D3) | TF Card SD_D3 (D3) | SDIO data 3. |
| GPIO38 | IO38 (42) | GPIO (WS2812) | WS2812B RGB LED data in | Drives the onboard RGB LED (data line). Timing-sensitive (requires RMT or precise timing)[4]. |
| GPIO39 | IO39 (43) | LCD_RST | ST7789 RESET pin | Active-low reset for LCD. (Resets display controller). |
| GPIO40 | IO40 (44) | LCD_SCLK (SCLK) | ST7789 SCL (serial clock) | SPI clock for LCD. High-speed signal (max peripheral SPI speed ~80 MHz, but LCD limited). |
| GPIO41 | IO41 (45) | LCD_DC | ST7789 D/C (data/command) | Data/ Command select for LCD SPI. |
| GPIO42 | IO42 (46) | LCD_CS | ST7789 CS (chip select) | Active-low chip-select for LCD SPI. |
| GPIO45 | IO45 (51) | LCD_DIN (MOSI) | ST7789 SDI (data in) | SPI MOSI line to LCD (master- |

| GPIO | SoC Pin | Function | Connected To | Notes/ Constraints |
|---|---|---|---|---|
| | | | | out-data). |
| GPIO46 | IO46 (52) | LCD_BL / USB_N | Q1 MOSFET gate (LCD backlight) / USB_N? | Drives LCD backlight MOSFET (via resistor R7 = 499 Ω). Also labeled USB_N in schematic (internal USB D–). |
| GPIO47 | IO47 (53) | *nHSPI? (USB_P) | WS2812 unused / USB_P? | Not used by peripherals, may be ESP32 USB D+. |
| EN (CHIP_PU) | EN (Chip) | RESET (chip enable) | RESET button (active low) | Tied to pull-up R1=10 kΩ to 3.3 V[5]. Press RESET to restart. |

- **Other GPIOs (IO2, IO4–IO13, IO19–IO37):** Most are brought to the header (see board overlay or [32]), and are free for user I/O (UART, SPI, I²C, PWM, ADC, etc.). None of these are used by core board functions, but beware of *strapping pins*: certain GPIOs (e.g. IO0) must have correct levels at reset for normal boot[2].
- **Voltage limits:** All GPIOs are 3.3 V tolerant. Do not exceed 3.3 V on inputs.
- **High-speed signals:** The LCD SPI lines (MOSI/SCLK/DC/CS) run at up to ~40 MHz for display refresh, requiring short trace runs. The SDIO lines (SD*, CMD) are also high-speed (up to 40 MHz) – the board adds 10 kΩ pull-ups on SD_D1–D3 and 0 Ω links for D0/D2 in the schematic, per typical SD card interface practice.

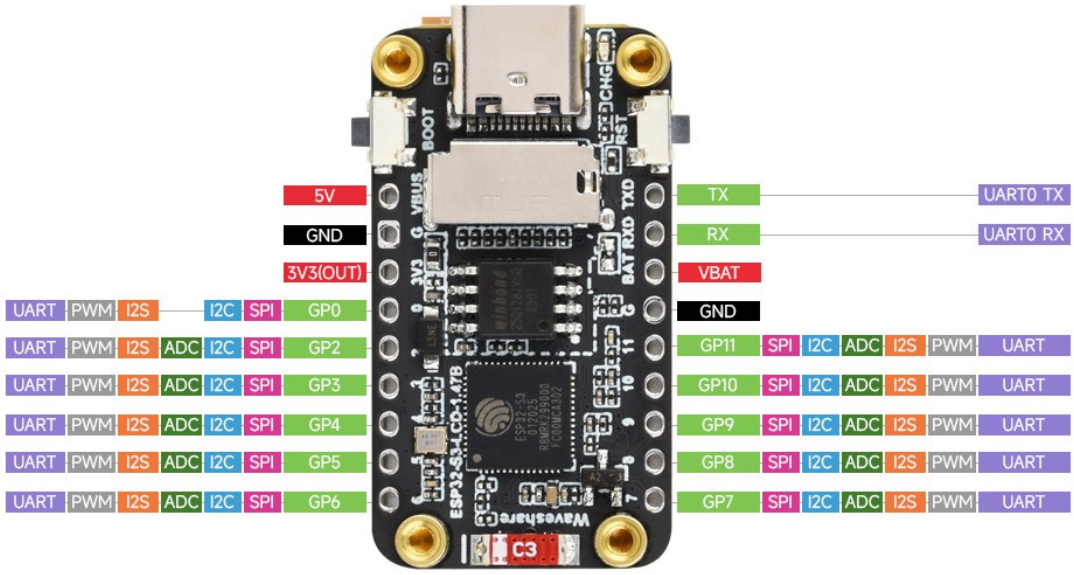


*Figure: Pinout of the ESP32-S3-LCD-1.47B board, showing 5V, GND, 3.3V rails, BOOT/RESET buttons, and GPIO header pins (GP0–GP11). Useful for reference during PCB wiring and debug.*

## Board Components Overview

- **ESP32-S3 SoC (ESP32-S3R8):** Dual-core 32-bit LX7 CPU, up to 240 MHz. Integrates 2.4 GHz Wi-Fi (802.11b/g/n, 40 MHz), Bluetooth 5.0 (LE/Mesh). On-chip memory: 512 KB SRAM, 384 KB ROM[1].
- **Flash (U4 – W25Q128JVSI):** 16 MB SPI flash (supports QSPI, up to ~104 MHz). Stores bootloader, firmware, partition table.
- **PSRAM (U3? not explicitly listed but onboard 8 MB):** External 8 MB QSPI PSRAM for frame buffer or data; connected to the SPI controller. High-speed, but require proper address line timing.
- **Display (LCD1 – 1.47″ ST7789 panel):** 172×320 pixel 262K-color IPS TFT, driven by ST7789 controller. Connected via 3-wire SPI (DIN, CLK, DC, CS, RST) plus a backlight LED. Backlight is an LED strip (Drives via transistor Q1). The backlight enable uses GPIO46 (via MOSFET Q1) with series R7=499Ω (for ~7 mA current with 3.3 V, per spec).
- **PMIC/LDO (U1 – ME6217C33M5G):** 3.3 V LDO regulator (800 mA max) fed from USB or battery. Provides stable 3.3 V rail. On input side (USB 5V) there's an input ferrite L1=24 nH and decoupling caps.
- **USB Interface:** USB Type-C receptacle (with CC resistors not shown) for power and data. The ESP32-S3's internal USB-Serial/JTAG lines (USB_DP, USB_DM) are connected directly to the USB port (no FTDI

chip). USB_VBUS (5 V) goes to charger and LDO input. On-chip pull-ups determine USB mode; note *USBOTG-4289* erratum: very early S3 chips had USB-OTG disabled by eFuse[6] (affects downloading from USB in boot mode on some boards).

- **Battery Charger (IC U2, likely a TP4056 or similar):** Circuit allows charging a 3.7 V Li-ion via USB. Battery voltage is regulated back to 3.3 V by the LDO. The battery connector ("system battery interface") is at the header. Battery presence and charge status LEDs (if any) are on board.
- **6-axis IMU (U? – QMI8658):** I²C sensor (3-axis accelerometer + gyro). Likely connected on I2C lines (one can assume GPIO17/SDA and GPIO18/SCL or similar, though the wiki doesn't explicitly list it). Used for motion detection, step-counting.
- **RGB LED (U5 – WS2812B):** Single WS2812 LED on board (needs precise single-wire timing). Controlled by GPIO38 via R19=10 kΩ.
- **Buttons:**
- **BOOT (Key1):** Connected to GPIO0 (strapping pin) with pull-up. Pressing enters download mode.
- **RESET (Key2):** Connected to EN (Chip_PU) through R1=10 kΩ to 3.3 V. Pressing resets the SoC. (R2=10 kΩ pulls BOOT high when not pressed, per schematic [1]).
- **TF Card Slot:** Standard microSD (TF) slot wired as SDIO. All four data lines (D0–D3), command (CMD), and clock (CLK) go to GPIO16–GPIO21, GPIO14, GPIO15 respectively[7]. The board provides pull-ups (10 kΩ) on D1–D3 and WP/CD lines (R12–R16 in schematic) to ensure idle high.
- **Misc:** A set of indicator LEDs (besides RGB) may be present, and an acrylic diffuser plate to enhance the RGB glow.

All board nets and passive components (decoupling caps on 3.3 V and flash Vcc, USB ESD diodes, LED current-limit resistors, etc.) are per standard design. The schematic (Waveshare) should be consulted for detailed net names.

## Risk Analysis

We categorize potential failure modes and risks as follows:

### ⚡ Power and Electrical Risks

- **Brownout / Voltage Drops:** Under heavy load (Wi-Fi TX, LCD refresh, SD card access), the 3.3 V rail may sag. If the ME6217 LDO approaches its current limit (~800 mA), its dropout voltage increases (risk if 5 V input dips). Low input (e.g. weak USB supply or low battery) could cause voltage regulators to drop out, triggering ESP32-S3 brownout resets.
- *Likelihood:* Moderate under maximum load or weak supply.

- *Impact:* Sudden resets, flash corruption, undefined behavior.
- *Root Cause:* Exceeding regulator capacity or USB port current limits.
- *Detection:* Use a multimeter or scope to check 3.3 V under load, enable brownout detector (BOD) in firmware.
- *Mitigation:* Use a robust 5 V source (≥1 A), add input decoupling (100 µF+ ceramic caps on VIN and battery), ensure firmware uses ESP32's brownout interrupt to reset gracefully. Possibly reduce CPU clock or disable power-hungry modules (BT) when not needed.

- **USB Instability / Overcurrent:** Rapid USB plug/unplug or faulty cables can cause spikes. Also, some early ESP32-S3 chips had USB-OTG disabled at boot (erratum **USBOTG-4289**) – meaning ESP32-S3 might not enter serial bootloader via USB until its eFuse is programmed[6]. On this board, USB "serial" is native, but SPI flash bootloader on old chips *may* not allow USB download.

- *Mitigation:* Keep firmware updated (IDF v5.x+ fixes USB issues), or use UART-to-USB booting as fallback. The esptool documentation notes pulling GPIO0 and GPIO46 low to enter bootloader[2]. Ensure BOOT button works.

- **Battery/Battery Charger Risks:** Charging electronics can overheat or fail. If the battery is shorted or reversed, fuse or protection must engage. Over-discharging the LiPo (below ~3.0 V) could cause undervoltage.

- *Mitigation:* Use only recommended batteries, and consider adding external fuse. Monitor battery voltage in firmware to warn at low level.

- **EFT/EMI:** USB and 5 V lines should have protection (ESD diodes, choke L1). Board likely has minimal filtering on USB. High-frequency activity (Wi-Fi TX) could inject noise into supply or ground.

- *Detection/Mitigation:* Using an oscilloscope to look for ringing; if excessive, add additional ferrite beads or caps. Proper grounding layout is key.

## Thermal Risks

- **ESP32-S3 Heating:** Under full wireless + CPU load (240 MHz + Wi-Fi TX), the chip can get very warm. Combined with limited copper area on dev board, thermal stress could cause throttling or damage.
- *Likelihood:* High if running heavy video updates on LCD + Wi-Fi concurrently for extended periods.
- *Impact:* CPU throttling or protective shutoff; reduced lifespan.
- *Mitigation:* Ensure good airflow. Use lower CPU frequency or light sleep between frames. In firmware, monitor `esp_get_temperature()` if

available. A small heatsink on the module (if unsoldered, but not on this dev board) could help.

- **LDO/Charger Heat:** The ME6217 LDO at 800 mA dissipation: Power loss = (Vin–Vout)×I. If powered from 5 V USB and 3.3 V out at high current, e.g. (5–3.3)×0.5A ≈ 0.85W dissipation, heating the LDO. The tiny SOT23-5 package may overheat if near max load. The LiPo charger chip (likely TP4056) also heats at high charge currents.
- *Mitigation:* Avoid taxing the board with >300 mA load off 3.3 V continuously. Use mid-power modes. Provide thermal relief to battery (wire it away from heat source).
- **Ambient/External:** If used in a hot environment or enclosure (e.g. inside a device), both the SoC and LDO could over-temp.
- *Detection:* Use an IR thermometer or thermal camera to spot hotspots.
- *Mitigation:* Add ventilation or de-rate workload.

## Signal Integrity and I/O Risks

- **High-Speed SPI (LCD) and SDIO Lines:** The LCD SPI (especially SCLK) and SDIO clock lines are high-speed (~10–40 MHz). Long traces or poor routing can cause signal reflections or jitter, leading to display glitches or SD card errors.
- *Mitigation:* Keep traces as short as possible. The board is small, but ensure decoupling near Flash/PSRAM (100 nF and 10 µF are present) as per schematic [1]. In firmware, avoid extreme SPI clock speeds if glitches occur; the ST7789 can be clocked moderately (e.g. 10–20 MHz) without artifacts.
- **USB D+/D– Lines:** These are differential; routing should be short and matched. The board's USB lines (DP/DN) must meet USB spec for connector operation. No HDMI/USB3, so less risk.
- *Detection:* Use scope to check USB signal quality during communication. If unstable, ensure Type-C wiring is correct (CC pins and E-markers if PD used).
- **GPIO Line Contention:** User attachments to GPIO (external devices) could back-drive pins. For example, if an external device drives a line as low/ high during boot, it could alter strap pin state or conflict.
- *Mitigation:* The bootstrapping guide warns internal pull-ups (e.g. 45 kΩ on GPIO0) exist[5]. External circuits on GPIO0 should have stronger pull-down to override, or simple avoid tying these pins low inadvertently.

- **EMC:** The on-board antenna (for Wi-Fi/BLE) needs space. Nearby metal or enclosures can detune it. For critical RF performance, ensure a clear area on PCB.
- *Mitigation:* Follow layout in eval board or add RF shielding if needed.

## Firmware / Boot Risks

- **Boot Strapping Errors:** As noted, **GPIO0** is a strap for boot mode[2]. If held low (e.g. shorted) at reset, the ESP32-S3 enters serial download instead of normal flash boot. Similarly, **GPIO46** (USB_D-) must not be floating high during boot if USB download is intended[8].
- *Likely Issue:* Accidentally leaving Boot button pressed at power-on will halt normal startup.
- *Mitigation:* Ensure proper pull-ups (10 kΩ present) and that user code doesn't mis-configure these pins early. Use the on-board BOOT button only when updating firmware.
- **PSRAM Initialization/Cache Bugs:** The S3's integrated flash/PSRAM interface has known errata. For example, "**CACHE-126**" (cache write-back error) means that under certain interrupt or multi-core conditions, cached writes may corrupt[9]. Also, some users report PSRAM crashes if initialization is incorrect (not in official errata but seen in forums[10]).
- *Impact:* Random crashes, memory corruption in graphics.
- *Mitigation:* Use updated ESP-IDF (v4.4.6+, v5.x) which includes workarounds (freezing cache during write-back, as Espressif notes[11]). In Arduino, use recent cores. Limit use of dual-core/multi-CPU writes to same PSRAM line concurrently.
- **LCD Controller Timing (Errata LCD-239):** The ESP32-S3 has a documented erratum **LCD-239**: when using the parallel RGB or I80 (8080-like) interface modes, certain clock divider settings cause misordered frames[12]. In 3-wire SPI mode (used here), this is less likely, but still relevant: ensure any LCD-related divider (e.g. pixel clock divisor) does not use the disallowed settings.
- *Mitigation:* Follow Espressif's workaround: do *not* set `LCD_CAM_LCD_CLK_EQU_SYSCLK = 1` in RGB mode; ensure `ahead_cycle > 2` if in I8080 mode[12]. (In our case, using SPI from software/LVGL means this detail is abstracted; just use standard library calls.) Confirm display doesn't misrender full frames.
- **USB Download Issue (Errata USBOTG-4289):** On pre-December 2022 ESP32-S3 chips, the EFUSE bit disabling USB download mode is set. If this board's ESP32-S3 batch is early, then attempting USB-based bootloader might fail.

- *Mitigation:* Use the BOOT button (GPIO0) and USB-Serial-JTAG console (which is the fallback per Espressif[6]). Espressif fixed this in later chips; verify chip's date code if necessary.

- **Cache and Flash Timing (Errata CACHE-126, ADC-183):** If an application uses SAR ADC2 with DMA, note **ADC-183**: the ADC2 DMA won't work[13]. The **CACHE-126** erratum can corrupt data if interrupts occur during cache write-back[9].

- *Mitigation:* Avoid ADC2+DMA (or use ADC1). Follow Espressif guidelines (use ADC calibration functions, skip ADC2 if needed). For cache, again use latest SDK with built-in freeze.

- **Other:** The analog power errata (**ANALOG-160**) warns not to disable certain analog blocks in sleep mode lest the chip be damaged[14]. This is unlikely in basic HMI apps, but avoid unusual deep-sleep analog power configurations.

Overall, known ESP32-S3 pitfalls must be addressed by using the latest Espressif libraries or applying recommended code workarounds. In firmware, enable watchdogs and watch for unexplained resets (which may hint at brownouts or errata manifestations).

## Test Plan and Validation

A formal test campaign is recommended. Key tests include:

- **Electrical Tests:**
- *Power rails:* Verify 3.3 V and 5 V rails under various conditions using a calibrated DVM or scope. Expect 3.3 V ±5% when idle and under peak load (Wi-Fi TX + display on).
    - *Pass:* 3.3 V stays between 3.1–3.4 V.
- *Current draw:* Measure current from USB (with a bench supply/multi-meter) in idle (deep sleep vs CPU idle) and active (Wi-Fi TX, display refresh).
    - *Pass:* Within expected ranges (e.g. <100 mA idle, <500 mA peak). No trips of USB port.
- *Brownout detector:* Deliberately lower 3.3 V (via adjustable supply) to trigger brownout. Ensure graceful reset if feature is enabled.
- *Button circuits:* Confirm BOOT and RESET buttons correctly pull pins low (check continuity to GND when pressed) and that pull-ups are present.
- *Voltage spikes:* Scope VBUS and Vbat during hot-plug or heavy load to ensure no overshoot.
- *Leakage/Short:* Check no DC shorts between rails (power off).

- **Functional Tests:**

- *Boot and Serial:* With no SD card, USB plug in, press RESET. Ensure bootloader prompt on serial console (115200 baud). Test BOOT button forces serial download mode (esptool etc.).
- *SPI Flash:* Run a flash reading demo. Validate entire flash (16 MB) is readable.
- *PSRAM:* Use a PSRAM allocation test (if available) to ensure full 8 MB is recognized.
- *LCD & RGB:* Run a display demo (e.g. LVGL or simple fill). Verify pixels on screen, no random glitches. Test backlight ON/OFF. Run RGB LED demo (rainbow cycles).
- *SD Card:* Insert a card, attempt mount (via Arduino or IDF). Read/write files to confirm reliability.
- *IMU:* Read accelerometer/gyro data via $I^2C$ (using sample code). Check sensible readings.
- *USB Functionality:* Transfer data over serial (USB CDC). Flash new firmware, erase flash, etc., to ensure USB works.
- *Charging:* With battery attached, measure charge current (expect ~500 mA typical). Monitor charge termination behavior.

- **Stress Tests:**

- *Max Load:* Run CPU at 240 MHz with both cores active, start continuous Wi-Fi UDP flooding, and simultaneously update the display (e.g. animated demo). Run for 1–2 hours.
    - *Pass:* No thermal shutdown; system remains responsive.
- *High-Temperature:* Operate board at elevated ambient (e.g. +60 °C). Verify stable operation.
- *Long-Run:* Run a graphic display loop (e.g. slideshow from SD card) continuously for 24+ hours. Check for memory leaks, crashes.
- *Electrical Stress:* Cycle USB power rapidly, cold/warm plugging, to check resiliency.
- *Battery Cycle:* Fully charge/discharge battery 10 times. Check charger behavior, battery thermals.

- **Edge-Case Tests:**

- *Cold Start vs. Warm Boot:* After deep-sleep or watchdog reset, ensure different wake reasons lead to expected behavior (e.g. proper LCD init each time).
- *SD Insertion/Removal Hot:* Insert or remove the TF card while powered (if supported by OS) to test for SDIO interrupt stability.
- *Brownout Emulation:* Gradually reduce supply and observe at what voltage resets occur, to calibrate the brownout threshold.
- *Serial Boot with Battery:* Press BOOT during power-on with only battery power (USB disconnected).

- *External I/O Overdrive:* Apply 5 V (mistakenly) to an input (safely) to test protection clamping (should not happen in normal use).

Each test should have clear Pass/Fail criteria (e.g. output not garbled, current under limits, no unexpected resets). Use a bench supply, oscilloscope, logic analyzer, and multimeter as appropriate.

## Firmware Boot Flow

Below is the boot sequence of the ESP32-S3 (aligns with Espressif's documentation[15]):

```
flowchart TD
    Reset[Power-on or RESET pressed]
    Reset --> Straps[Check GPIO straps (BOOT, etc.)]
    Straps -->|GPIO0 low| Bootloader_ROM(ROM Bootloader)
    Straps -->|GPIO0 high| Bootloader_ROM
    Bootloader_ROM --> Load2nd[Load 2nd-stage Bootloader from Flash
(offset 0x0)]
    Load2nd --> ParsePT[Parse Partition Table]
    ParsePT --> LoadApp[Load App Image from Flash]
    LoadApp --> SystemInit[Initialize peripherals (GPIO, clocks,
etc.)]
    SystemInit --> AppMain[app_main()]
    AppMain --> LCD_Init[Init LCD (SPI, reset, backlight)]
    LCD_Init --> Run[Run Application (GUI, WiFi tasks, etc.)]
```

1. **Reset / Power-On:** On power or reset, the PRO CPU starts from mask-ROM (unmodifiable). The ESP32-S3 reads *strapping pins* (e.g. `GPIO_STRAP_REG` checks GPIO0, GPIO46, others) to determine mode[16] [2].
2. **Bootloader (ROM):** The first-stage ROM bootloader executes. If GPIO0 is held low (BOOT pressed), it enters serial download mode; otherwise it proceeds normally. In normal boot, the ROM loader configures the flash and then jumps to the second-stage bootloader in SPI flash (at offset 0x0)[15].
3. **Second-Stage Bootloader:** The second-stage (in SPI Flash) verifies signatures (if secure boot), loads the partition table, and loads the main app partitions (e.g. factory app) into memory. It then starts the App CPU.
4. **Application Startup:** The ESP-IDF runtime initializes (setting CPU clock, memory, etc.), then calls `app_main()`. At this point Wi-Fi/BLE and other drivers can be initialized.
5. **Board Initialization:** In `app_main` (or prior), firmware must reconfigure pins. For instance, it will set GPIO39–GPIO46 for the LCD SPI, toggle the LCD_RST line to reset the display, initialize SPI bus, and set LCD_BL (GPIO46) as output to light the backlight. It will also initialize the IMU (enable I²C), mount the SD card filesystem, etc.

6. **Run:** Finally, the main application (UI task) runs, updating the display and handling sensors and network.

This flow ensures that on every reboot the peripherals (LCD, SD, IMU) are properly set up. Pressing RESET simply repeats this sequence. Pressing the BOOT button at reset diverts the flow to USB serial download (handled by ROM).

## Improvement Recommendations

- **Power Supply:** If frequent brownouts occur under load, consider a higher-current 3.3 V regulator or adding a second regulator for peripherals. Increasing input capacitance (e.g. an extra 100 µF on 5 V) can buffer surges.
- **Decoupling:** Verify all high-speed chips (PSRAM, Flash, IMU, ST7789) have close-by decoupling caps (0.1 µF + 10 µF) per datasheet. Add additional 0.1 µF on regulator outputs if missing.
- **Signal Integrity:** For long cables or interference-sensitive use, add series resistors (15–22Ω) on SPI clock/MOSI lines, and small series 47Ω on USB D+/D– to damp.
- **Firmware:** Always use the latest ESP-IDF or Arduino core. Enable `CONFIG_SPIRAM_SUPPORT` properly. Use the errata workarounds (e.g. `esp_rom_intr_set_level(3)` or cache freeze functions during Wi-Fi or DMA bursts). Limit LCD refresh rate to avoid timing issues.
- **Flash Reliability:** Implement wear leveling/CRC in storage, to mitigate any rare flash write failure.
- **Thermal:** If the board will be enclosed, consider adding a small heat sink or copper pour under the ESP32-S3 module (if possible). Use PWM at moderate frequency for backlight to minimize power.
- **Mechanical:** The LCD is soldered on top of the board. Ensure not to flex it; maybe add a protective front cover.

No catastrophic issues were identified, but following strict engineering validation (per the test plan above) is crucial for deploying in a product. All findings should be verified by lab tests.

**Sources:** Official Waveshare documentation and schematics[1][7], and Espressif datasheets/errata and programming guides[15][12][2]. Any data not found in these is marked or based on typical design practice.

---

[1] [3] [4] [7] ESP32-S3-LCD-1.47B - Waveshare Wiki

https://www.waveshare.com/wiki/ESP32-S3-LCD-1.47B?srsltid=AfmBOoo-cwnZQ6wCUQbE0KiDpr8pUc2iCS9zREys0FfUXPa5fLe7PAU8

[2] [5] [8] Boot Mode Selection - ESP32-S3 - — esptool latest documentation

https://docs.espressif.com/projects/esptool/en/latest/esp32s3/advanced-topics/boot-mode-selection.html

[6] [9] [11] [12] [13] [14] All Errata Descriptions - ESP32-S3 - — ESP Chip Errata latest documentation

https://docs.espressif.com/projects/esp-chip-errata/en/latest/esp32s3/03-errata-description/index.html

[10] ESP32S3: PSRAM crashes #816 - lovyan03/LovyanGFX - GitHub

https://github.com/lovyan03/LovyanGFX/issues/816

[15] [16] Application Startup Flow - ESP32-S3 - — ESP-IDF Programming Guide v6.0 documentation

https://docs.espressif.com/projects/esp-idf/en/stable/esp32s3/api-guides/startup.html